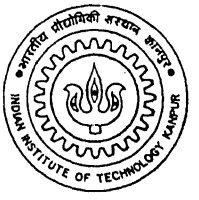
Geometric Modeling of Patterns

A Thesis Submitted
in Partial Fulfillment of the Requirements
for the Degree of
Master of Technology

by

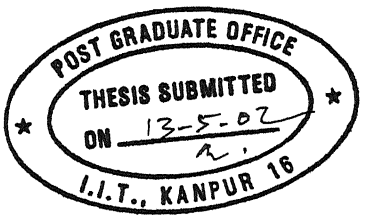
Kedar S. Patil



to the

Department of Computer Science & Engineering
Indian Institute of Technology, Kanpur

May, 2002



Certificate

This is to certify that the work contained in the thesis entitled "*Geometric Modeling of Patterns*", by *Kedar S. Patil*, has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

May, 2002

(Dr. Sanjay G. Dhande)
Department of Computer Science & Engineering,
Indian Institute of Technology,
Kanpur.

Abstract

A pattern is an orderly arrangement of objects in space. Such order is inherent in natural as well as man-made objects. Artists have been using patterns in art and architectural design for ages. Designing patterns is a tedious process involving skill, training, patience, aesthetic sense and artistic imagination.

In this work we propose a geometric model for two-dimensional patterns. This model is meant for incorporation into design software that can be used by persons having minimal artistic skills.

The proposed model is hierarchic and represents pattern as a tree. We have implemented the model in a program that reads in a shape description, builds a tree of shapes and renders the pattern. A graphical user interface is also provided for interactive editing of shape descriptions.

Acknowledgements

I dedicate this work to all known and unknown entities that have shaped my life.

I thank Dr. Sanjay G. Dhande for his guidance and encouragement during this work. I am still amazed that every talk with him gives me something new to explore.

Thanks are due to all faculty and staff members at the Department of Computer Science and Engineering at IIT Kanpur. I am also grateful to all individuals associated with the CAD Project Laboratory at IIT Kanpur, for their care and support.

I thank all my batch-mates at CSE Department, IIT Kanpur for bearing my company for two long years. I will always miss this small circle of closely knit friends. I am particularly grateful to Neeraj for the useful discussions and the daily exchanges of "what new did we learn today" reports.

I am grateful to my father and mother who have brought up a troublesome kid like me with their infinite love and patience. My little sister deserves a word of praise for being a nice and caring companion.

Finally, I would like to thank the world-wide programmer community for producing good quality free software and equally good documentation that I have been using till now.

– Kedar Patil

Contents

List of Figures

Chapter 1

Introduction

1.1 Patterns

All natural and man-made things have an order built inside them. Whether it be a flower or the human form or a pretty design on cloth, all exhibit some kind of proportion, pattern and unity that makes them pleasing to our eyes. Abstract things like language and music also have some inherent order or pattern that provides a pleasant feeling to our senses.

Patterns in nature are formed by biological systems (plants, animals and other life-forms) or by inanimate things (crystal lattices, chemical and physical reactions). Biological patterns are said to have *Dynamic Symmetry* as opposed to *Static Symmetry* in inanimate patterns. Man-made patterns are usually simplified versions of natural patterns, copied knowingly or unknowingly by artists, and passed on as a tradition from one generation to the next.

Patterns are essential component of our day-to-day life. We can see them in art, architecture, and items of daily use such as textiles, utensils, furniture and decorative objects.

1.2 Design and Analysis of Man-made Patterns

Artists use their imagination and artistic skills to design patterns. For producing such designs artists need special talent, training, patience and aesthetic sense. Designing is

therefore a specialized and time consuming endeavor.

The complexity of designs generated by artists is usually very high. It requires considerable mathematical analysis to discover order in such designs and encode them in mathematical descriptions. Moreover, such analysis is usually specific to a particular design (or a small class of designs) and is difficult to generalize.

1.3 Computer Generated Patterns

Computer can be used as a design tool to help artists realize their designs by freeing the artist from unnecessarily tedious or repetitive tasks. This allows the artist to work on the design idea rather than its implementation. Such tools are common in use now and are known as Computer Aided Design (CAD) tools. These tools allow various degrees of freedom to designers helping them to manipulate a set of fundamental building blocks and to arrange the blocks into some definite order to form a pattern.

These tools are very effective in considerably reducing the effort and time required for designing. But these are not a substitute for artistic skills. The user of such tools needs to have some kind of artistic talent to utilize them efficiently and produce pleasing designs. If we could have a tool that has "artistic sense" built into it, then it would be possible for a non-artist person to design and experiment with patterns. Such sense can be put into a computer tool if we can encode the design in a format that is understood by a computer as something more than just a bunch of shapes put together.

1.4 Objectives and Scope of Present Work

An average person does not usually possess artistic skills. But he/she surely has some imagination as to what will be pleasing to look at. This leads us to believe that it will be helpful if we provide such a person with a computer tool to enable him/her to produce his/her own designs. To build such a tool, it is desirable to have a computable model of a class of designs that are widely used and appreciated. Therefore the present work is geared towards design of patterns that are used in textiles, perhaps more prominent in embroidered works. Such patterns are usually floral patterns, displaying dynamic symmetry.

These patterns are essentially two-dimensional patterns.

In this report, we propose a hierarchical computable model for such patterns. Then we describe our implementation of that model. Finally we study the results of the implementation, that is the visual output obtained from the program.

1.5 Organization of Present Report

This report is organized into following parts, each forming the contents of a chapter, starting from second chapter onwards.

- Basic concepts for understanding patterns.
- A description of the pattern model.
- Implementation and results.
- Conclusion and suggestions for future work.

Chapter 2

Background

2.1 Patterns and Symmetry

Symmetry is an intrinsic property of a mathematical object which causes it to remain invariant under certain classes of transformations such as rotation, reflection, inversion, or more abstract operations. [13]

Symmetry deals with similarity of shapes. A shape is said to be symmetric if there is a transformation that when applied to the shape brings it back to its original form. A common example of symmetry is mirroring. Mirrored shapes "look like" each other – mirroring one of them again, makes them coincide.

Symmetry is identified with transformation that leads to it. Here we consider only those transformations that preserve distances and angles between entities that make up a shape. Such transformations are known as *isometries*. They can be thought of as *rigid body motions*. There are four isometries (and their linear combinations) that lead to symmetry and patterns in a two-dimensional plane.

1. **Translation**

   Translation is displacement in a particular direction. It is completely specified by two parameters dx and dy, which specify the displacement along the two principle axes.

2. **Rotation**

   Rotation is specified by an angle and a point to rotate about. Rotations about origin

are common. In general rotations can be by any angle, but only few angles are actually used to produce patterns. These angles are 60°, 90°, 120°, 180°. The *order of a rotation* is defined as the number of times that particular rotation needs to be applied to bring the plane back in its original position. Therefore, rotation by 60° is of order 6, 90° is order 4, 120° is order 3, and 180° is order 2. Order 2 rotation, or rotation by 180° degrees is also termed as a *half-turn*.

3. **Reflection**

   Reflection needs a line called as axis of reflection. To reflect a point in a given axis of reflection, is to exchange a point with another point on the other side of the axis. The line passing through a point and its reflection is perpendicular to the axis of reflection. Reflection is also known as *mirroring* due to similar behavior exhibited by a mirror.

4. **Glide Reflection**

   Glide reflection is reflection in an axis followed by translation along that axis.

Figure 2.1 shows patterns formed by these isometries.

A pattern is a regular arrangement of objects in space. For the purpose of this report, we consider patterns made by arrangement of two dimensional objects lying in a two dimensional plane. A pattern is identified with the type of symmetries that leads to it. For example, the figure 2.2 depicts a pattern generated by translation in two different directions.

2.2 Static Symmetry and Dynamic Symmetry

Symmetry is classified into two types — *Static Symmetry* as observed in geometrical patterns, and *Dynamic Symmetry* as observed in growth patterns in nature.

Static symmetry involves transformations described above. It is seen as work of art, architecture, design or in geometrical arrangements in nature such as crystals.

Dynamic symmetry refers to the symmetry of proportional growth. One example is similarity of increasing proportions, as depicted in figure 2.3. Starting with a rectangle, a square is drawn on the larger side. The result is a bigger rectangle for which the procedure

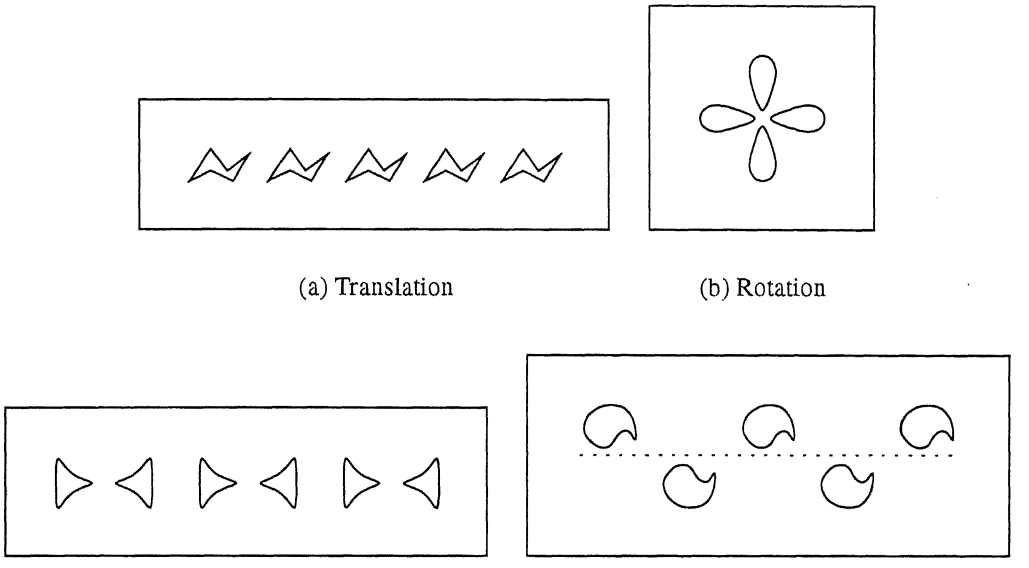


Figure 2.1: The Four Isometries

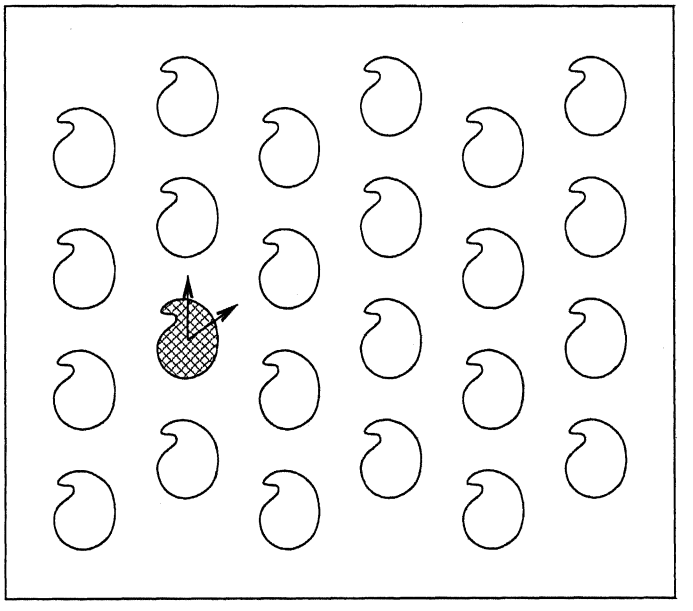


Figure 2.2: Pattern Generated by Translation in Two Directions

is repeated. Arcs of circle are drawn in each square. This gives rise to an approximation of a logarithmic spiral.

Such spirals are formed in nature due to proportional growth of plants and animals. Each stage of growth bears a fixed proportion with the previous stage, giving rise to the logarithmic spirals.

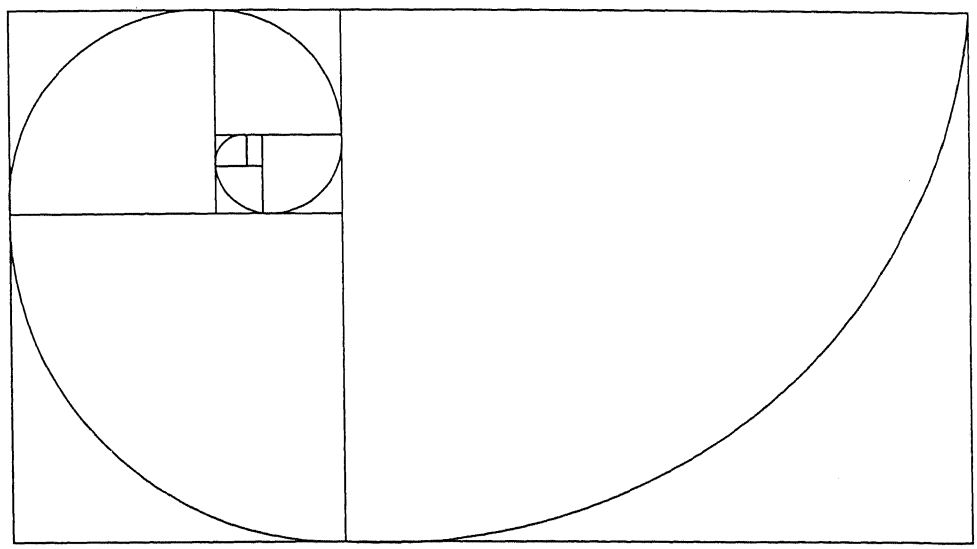


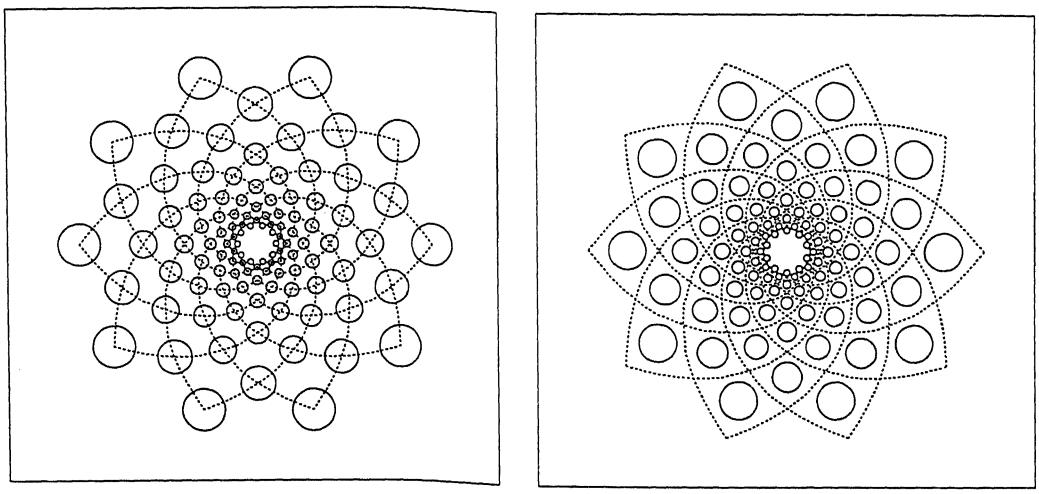
Figure 2.3: Dynamic Symmetry: Similarity of Increasing Proportions

Dynamic symmetry is observed in plants (see figure 2.4) , animals (e.g Fish and shelled organisms) and human form. Principles of dynamic symmetry are in use knowingly or unknowingly in architecture, pottery and handicrafts for thousands of years. [4]

2.3 Shape Grammar

Shape Grammars provide a means for the recursive specifications of shapes and are similar to phrase structure grammars developed by Chomsky. The alphabet of shape grammar comprises of shapes and the language generated is language of shapes, notably a language of patterns. [7]

A shape grammar consists of four entities:



(a) Eyes on Peacock's Tail

(b) Center of Daisy Flower

Figure 2.4: Dynamic Symmetry: Intersecting Spirals

1. Set of **Terminal Shapes** (T)

   Terminal shapes are shapes that make the target shape. Terminals are analogous to alphabets of language. Every shape in the language described by the shape grammar is composed entirely of terminals.

2. Set of **Marker Shapes** (M)

   The purpose of marker shapes is to distinguish between terminal shapes and non-terminal shapes, which appear on the left hand side of rules. Without markers, terminals and non-terminals may be indistinguishable.

3. Set of **Rules** (R)

   Each rule is of the form $\alpha \rightarrow \beta$ where α comprises of one or more shapes out of which at least one is a non-terminal and β comprises of one or more shapes, each shape being either terminal or non-terminal. Starting with initial shape, rules are successively applied to the current shape to transform it to another set of shapes. Usually there is a rule that erases markers, so that the process of applying rules stops there. This is better explained by an example, see figure 2.5.

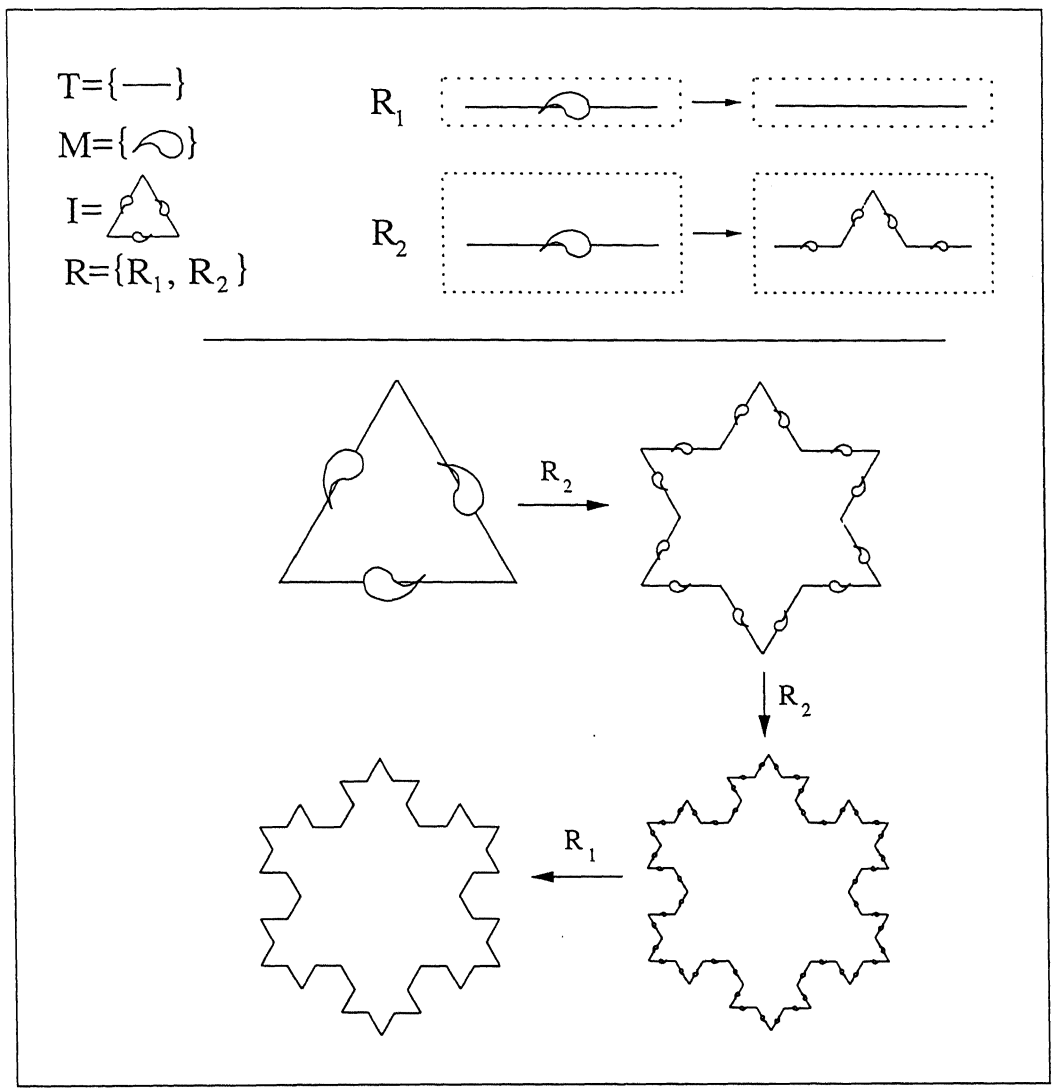


Figure 2.5: Shape Grammar for von Koch Snowflake

4. **Initial Shape** (I)

   Initial shape serves as the initial template upon which rules are applied to generate a shape in the language specified by the shape grammar.

Figure 2.5 demonstrates a simple shape grammar comprising of two rules. The first rule simply erases a marker from a line segment. The second replaces a line segment with a marker with four lines of one-third the length of original segment, forming a kink in the middle. Note that all four segments have markers embedded in them. This makes them suitable for further application of the grammar rules.

Starting with the initial shape, which is a triangle formed by three segments, we apply the second rule to each segment to get a star-like formation. Same rule is applied again to that formation to get a finer version. Finally the first rule is applied thereby preventing any further rule application. The result is the second iteration of von Koch snowflake. As many iterations as required can be applied to the initial shape to get more detailed pattern. That is the reason for shape grammars being defined as *recursive* specification of shapes.

Although introduced as a means of specifying abstract shapes, shape grammars have been applied to other kinds of design problems, especially architectural designs and product designs. Notable examples are that of Shape Grammar for Moghul Gardens and Coffee Machine Shape Grammar. A particularly fascinating example is Harley-Davidson Shape Grammar which generates designs for motorcycles.

2.4 Minkowski Sum

Minkowski sum of two sets (of points) $\mathcal{A}$ and $\mathcal{B}$ is defined as:

$\mathcal{A} \oplus \mathcal{B} = \{a + b \mid a \in \mathcal{A} \text{ and } b \in \mathcal{B}\}$

where $a + b$ is the vector sum of position vectors of point a and point b.

Figure 2.6 illustrates Minkowski Sum of a square and a triangle. The sum is depicted as convex hull of the resulting points. The sum can be visualized as the triangle being placed at each vertex of the square and convex hull of the result being taken.

If we use the notation $a + \mathcal{B}$ to stand for the set $\{a + b | b \in \mathcal{B}\}$ then Minkowski sum can also be expressed as:

$\mathcal{A} \oplus \mathcal{B} = \cup(a + \mathcal{B})$ taken over all $a \in \mathcal{A}$.

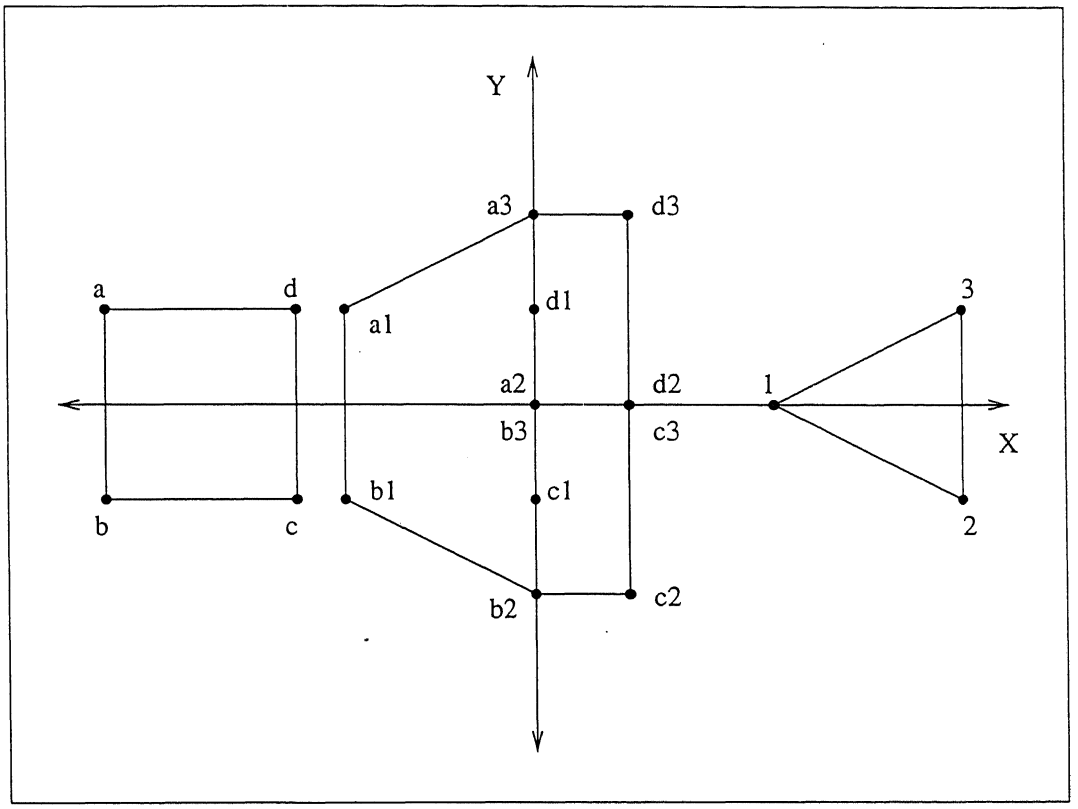


Figure 2.6: Minkowski Sum of a Square and a Triangle:
Vertex labels of the sum are formed using labels of points contributing to that vertex. For example, label ***a1*** *indicates that it is a sum of vertex* ***a*** *and vertex* ***1****.*

This interpretation is of special interest to us. It can be viewed as a result of an operator applied to the set $\mathcal{B}$ for each element of set $\mathcal{A}$. In the above definition the operator is vector sum (+), but it can be replaced by another operation such as transformation. This is one of the key concepts used in present work and will be explored in the next chapter.

Chapter 3

Geometrical Model for Patterns

The model we propose is *Hierarchic* and makes abundant use of *Primitive Instancing*.

Hierarchy allows us to deal with enormous amount of details by dividing the details into several levels which can be tackled independently. A pattern can be represented hierarchically as a rooted tree with nodes representing components of the pattern. Children of a node represent substructures within the structure that is represented by the parent node.

Primitive instancing allows us to reuse the same primitive (shape used to construct a pattern) at multiple locations in the pattern without duplicating the description of the shape. The primitive has a fixed description but the individual instances carry other variable parameter information such as size, position and orientation. This saves tremendous amount of space in representation of complex patterns where there are large number of similar components. Also, modifying a primitive changes all its instances in the pattern.

Consider for example a simple pattern consisting of 10 petals repeated around a circle. If we add 10 separate descriptions of petal in our model, it will waste space since each petal is essentially similar to other, just the positions and orientations differ. The solution is to add 10 *instances* of petal to the model, with position and orientation details for each petal. Moreover, change in petal shape changes every petal instance. In the former case, editing each petal would be required. Thus, editing a pattern is reduced to editing the primitives only.

The model is based on Minkowski Sum. As pointed out in the previous chapter, Minkowski sum can be treated as the aggregate result of an operator applied to a set $\mathcal{B}$ for

each element of another set $\mathcal{A}$. We use affine transformation as the operator and transform $\mathcal{B}$ by using elements of $\mathcal{A}$ which is a set of transformation matrices. By changing the set of transformation matrices, we can change the pattern in which shape $\mathcal{B}$ is arranged.

3.1 The Pattern Model

The model is defined in terms of a single entity called as *Shape*. Shapes interact with each other forming a hierarchic structure called as *Pattern Tree*.

3.2 Shape

A shape

- can transform itself.
- can draw itself. We refer to this as *rendering*.
- can provide a list of *anchors* when requested. An anchor is a 3×3 two-dimensional homogeneous transformation matrix.

These simple properties are sufficient for pattern formation. Shapes are further classified into following types

- Simple Shape
- Compound Shape

3.2.1 Simple Shape

A simple shape has following structure.

1. **A Geometrical Description**
   This description determines how the shape is to be rendered. It may consist of points, lines, curves, and local transformations. It may be empty. This is to allow

shapes that are not meant to be rendered but to be used as templates for positioning other shapes. The exact representation of this description is implementation dependent.

2. **A List of Anchors**
   Each anchor in the list is a 3×3 two-dimensional homogeneous transformation matrix. Ideally, elements of this list should relate to the shape in some way, such as anchors that correspond to equidistant points on the periphery of the shape. The anchor list may be empty.

3. **A Local Coordinate System**
   The shape description and the anchors use a coordinate system that is local to the shape. All transformations done by the shape are in this coordinate system.

4. **A Set of Shape Attributes**
   Attributes are used while rendering the shape. Typically these include line thickness, line color, fill color, fill pattern, etc. There may be flags for things like visibility of the shape. The exact details of attributes and the associated semantics is left to the implementation. The only requirement is that the attribute changes should be restricted to that shape only.

3.2.2 Compound Shape

A compound shape is a list of references (pointers) to other shapes and some flags to specify the semantics of utilizing the list.

Depending on the flags compound shape behaves as one of the following

- **Aggregate**
  Aggregate Compound Shape is just a collection of shapes. All shapes are rendered one by one whenever request for rendering the compound shape arrives.

- **Pattern**
  Pattern treats the shape list in a special way. The first shape is treated as *Base-Shape* and the rest are treated as *Repeat-Shapes*. While rendering, anchor-list is requested

from base-shape and repeat-shapes are rendered, one repeat-shape per anchor. If the number of anchors is more than the number of repeat-shapes, the repeat-list is treated as a circular list.

3.3 Pattern Tree

To form a pattern, shapes are declared. Some of them are simple and some are compound. This leads to a hierarchy of shapes which can be represented as a *Pattern Tree*. A sample pattern tree is shown in figure 3.1

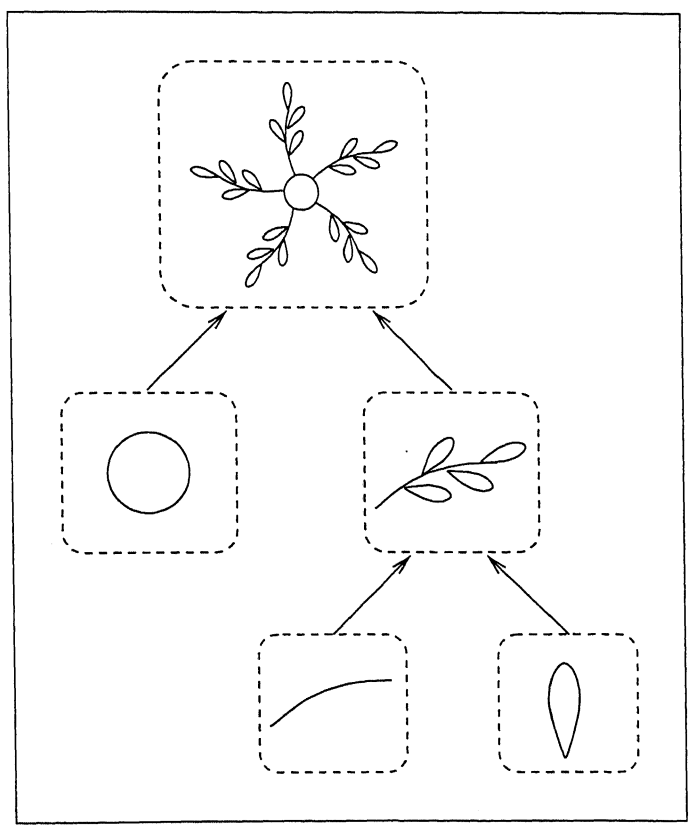


Figure 3.1: Pattern Tree

The internal nodes represent compound shapes and leaf nodes represent simple shapes.

3.4 Rendering

3.4.1 Rendering Simple Shape

A simple shape renders itself using its geometric description and attributes. Details depend on the implementation.

3.4.2 Rendering Compound Shape

An aggregate compound shape is renders itself by asking all its sub-shapes to render themselves one by one.

A pattern compound renders itself by performing following steps

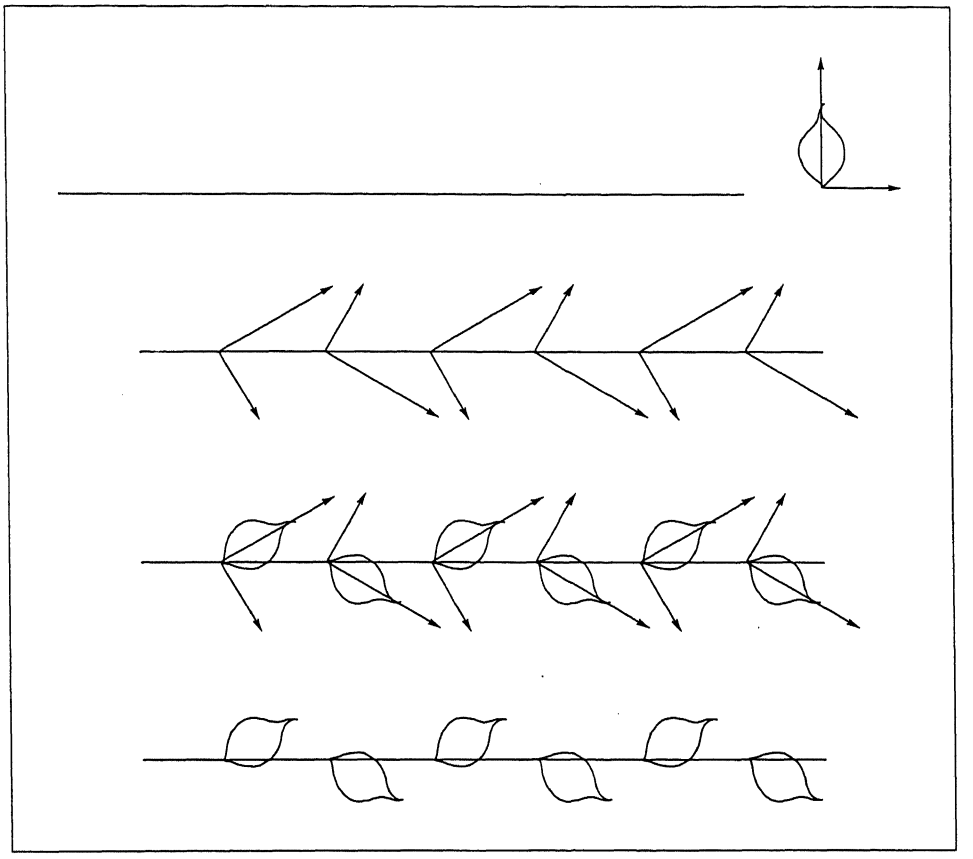
- Get anchor list from base-shape. Consider the repeat-shapes as being in a circular list.
- For each anchor in anchor list, do the following
  - Transform the current coordinate system using the anchor
  - Pick the next repeat-shape, ask it to render itself.
  - Undo the transformation done to the coordinate system.

Thus, rendering is a recursive process. It will traverse the pattern tree using depth first search and render each node. Figure 3.2 demonstrates the rendering of a compound pattern shape.

3.5 Managing the Anchor List

The anchor list is the most crucial part in the pattern model. The orderly arrangement of shapes to form a pattern is the result of carefully chosen entries in the anchor list. All modifications in the arrangement of shapes in a pattern are essentially manipulations on the anchor list.

As stated earlier, anchors are matrices. Each matrix is a three by three matrix which serves as a homogeneous transformation matrix for a two-dimensional coordinate system.



Compound shape consisting of a line and a leaf. The anchors are shown as a coordinate system with a long y-axis and a short x-axis. For each anchor, the leaf is transformed so that its coordinate system coincides with the anchor coordinate system and then rendered.

Figure 3.2: Rendering a Compound Pattern Shape

A simple shape maintains an explicit anchor list. It may be derived from the simple shape by some operation on the geometrical representation of the shape. Equidistant points on the periphery of the shape with direction tangent to the periphery is a common example. The anchor list may arbitrarily obtained using any algorithm, or explicitly specified. This allows for patterns such as grids or polygons or more exquisite arrangements.

A compound shape may maintain its own explicit anchor list. But more commonly, it derives its anchors from the sub-shapes it contains. Copying anchors of first sub-shape is a good choice. The details of such derivation and whether to allow explicit anchor list in a compound shape, are left to the implementation.

Chapter 4

Implementation

4.1 Language and Tools Used

We have implemented the pattern tree using the C++ language. C++ was chosen because the pattern model is object-oriented by nature. Each shape can be considered as an object having methods that render it and that return the anchor list.

The NURBS++ library is used for curve manipulations. NURBS++ is a free C++ implementation of NURBS curves providing a vast array of operations on the curves.

Adobe PostScript format is used for the output. PostScript is a powerful page description language that can be used to produce high quality printed outputs using a laser printer. The language provides powerful primitives for two-dimensional transformations. It also allows definition of procedures. This makes the task of producing output simpler.

Graphical User Interface (GUI) is also provided for editing the pattern tree. It uses GTK- -, the C++ version of GTK, the GIMP Tool-Kit, for the user interface elements also known as widgets. GTK- - was chosen for its well defined API interface and ease of use.

Flex and Bison are used to write parser for the shape description file format.

4.2 Shape Description Format (SDF)

We have implemented the pattern model using a *Shape Description Format.* It is an ASCII file format that describes simple shapes in terms of interpolated curves and compound shapes in terms of simple shapes. Interpolated curves were chosen since they can

represent the common primitives in floral patterns easily. Also the points for interpolation can be determined easily by drawing the primitives on a graph paper. Such primitives include curved vines, petal outlines, leaf outlines, etc.

The shape description format consists of following entities

- **Global Attributes and Parameters**

  This section contains global attributes for rendering shapes. These are used if the shape themselves do not have their own attributes. Also some transformations are allowed to allow overall effects on the pattern, such as adjusting its position on the output page, scaling entire pattern, etc. Page size can also be specified.

- **Simple Shape Description**

  This consists of geometric description of curves that make the shape. The curve is specified by means of a list of points lying on that curve. The points are interpolated as and when needed to get a NURBS curve. The list of anchor points can also be specified.

  Optional attributes can be specified to alter line color, line thickness and fill color of the shape.

- **Compound Shape Description**

  Compound shapes are specified as lists of other shapes. Other shapes are referred to by their shape-id which is unique to every shape. Optional flags can be specified to alter the behavior of the compound shape as discussed in the pattern model.

- **Render Requests**

  Render request is simply a directive to the program to add the shape to a list of shapes. The list is used for rendering when the pattern tree is ready.

These entities can occur in any order in the file. The only constraint is that a shape should be declared before using it in compound shapes or render requests.

.3 SDF file format

An outline of the SDF file format is as follows. Exact definition of the format can be ound in appendix B. Note that the format is quite verbose to facilitate manual editing. omments are started with the character '#' and extend upto the end of line. The first ne is a special comment to specify the version of the file format. This is to allow fu- ire extensions to the file format while maintaining backward compatibility for older file ormats.

```
!SDF-1.0

lobal

    transform  {  }  # transformations
    attributes {  }  # color, line thickness, etc
    output     {  }  # output options


hape <shape-id>

    simple      # shape type
    points
    {
        {x1,y1}, {x2,y2} [, ... ]
    }
    anchors
    {
        grid { ... }     # inbuilt procedures to create anchors
        transform { ... }   # explicit transformations
    }
    attributes { ... }
    transform { ... }
```

```
}

shape <shape-id>
{
    type { compound }     # shape type
    flag { aggregate }    # "aggregate" or "pattern" compound
    shapes { <shape-id1>, <shape-id2> [, ... ] }
    transform { ... }
}

render
{
    <shape-id1> [, ... ]
}
```

4.4 "pattern" - The Pattern Generator

The SDF file is used by the program "pattern". Pattern is a command-line non-interactive parser program takes the name of SDF file as a command-line parameter and produces a PostScript description of the pattern to a file whose name can optionally be specified on the command-line. Otherwise it outputs it to standard output. The PostScript output can be fed to a laser printer to get a hard-copy or it can be viewed using programs like gv (ghostscript). It can also be converted to popular image file formats using ghostscript to use it on the World Wide Web.

4.5 Working of the Program

The program builds table of shapes by adding shapes to it in the order as they are described in the SDF file. It takes care that shapes are described before they are used to describe other shapes. It also reports any errors that may be present in the SDF file.

The program also builds a render list that includes references of shapes that are requested to be rendered. After reading the whole SDF file, the program proceeds to render all shapes in the render list one by one.

To render a shape, its PostScript output method is called. Each shape outputs a PostScript procedure that renders it. Compound shapes call the PostScript procedures of their sub-shapes after transforming the coordinate system by required transformation matrices and undo the transformation after the call. The entire combined output of all shapes can be treated as a PostScript file which can be viewed or printed.

The program takes care of other details in the PostScript file, such as Document Structuring Comments (DSC) headers and footers in the PostScript flie.

4.6 "gpattern" - The Graphical User Interface

A graphical user interface named "gpattern" is provided to interactively edit the SDF file.

The interface shows the pattern as an editable tree structure on the left side. Right side consists of a panel which displays the selected shape at the top and the editing panel at the bottom.

Each node of the tree represents a shape. A shapes can be edited by selecting the shape by mouse which will bring up its image and editing panel on the right hand side. The editing panel consists of variety of controls and input boxes through which every editable parameter of the shape can be changed.

Points on the curve can be edited by dragging them. Points can be added to or deleted from the list.

An anchor is represented as arrow whose base lies at the point represented by the anchor, its orientation specifies the direction and its length is proportional to the scaling factor. Anchors can be edited by dragging the arrow base to change its position, or by dragging the arrow tip to change the direction and/or scale.

Shapes can be selected for rendering by using check-boxes located next to a node in the tree structure.

Most of the editing procedures can be done by specifying exact numbers in the input

boxes provided on the editing panel. This allows finer control over the shape parameters.

The menu bar at the top contains options to load/save the tree to/from a SDF files, and to render the selected shapes.

4.7 Results

We tried the software to generate sample patterns that demonstrate the four basic isometries with the results in figure 4.1.

An example of a complicated pattern is shown in figure 4.2.

More sample outputs can be found in appendix B.

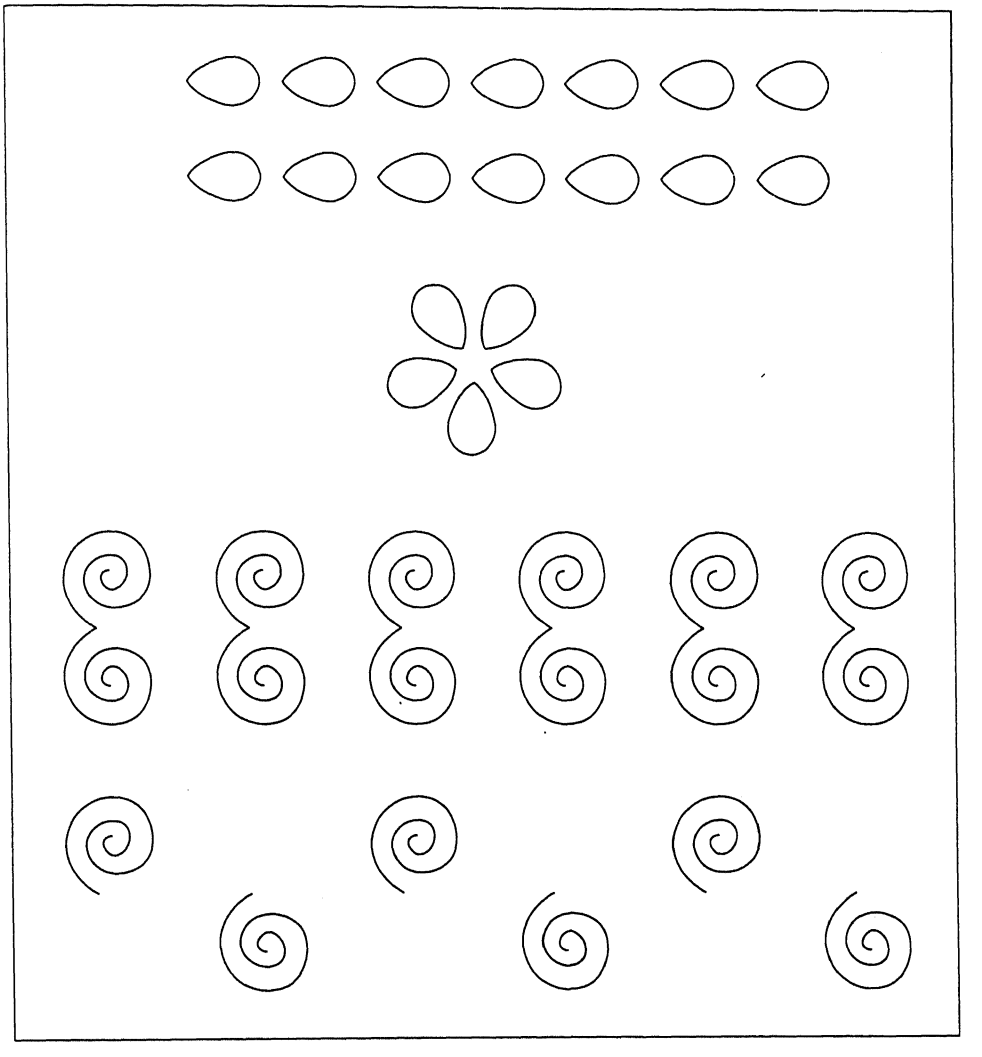


Figure 4.1: The Four Isometries produced using *pattern*
(Translation, Rotation, Reflection and Glide Reflection respectively from top to bottom)

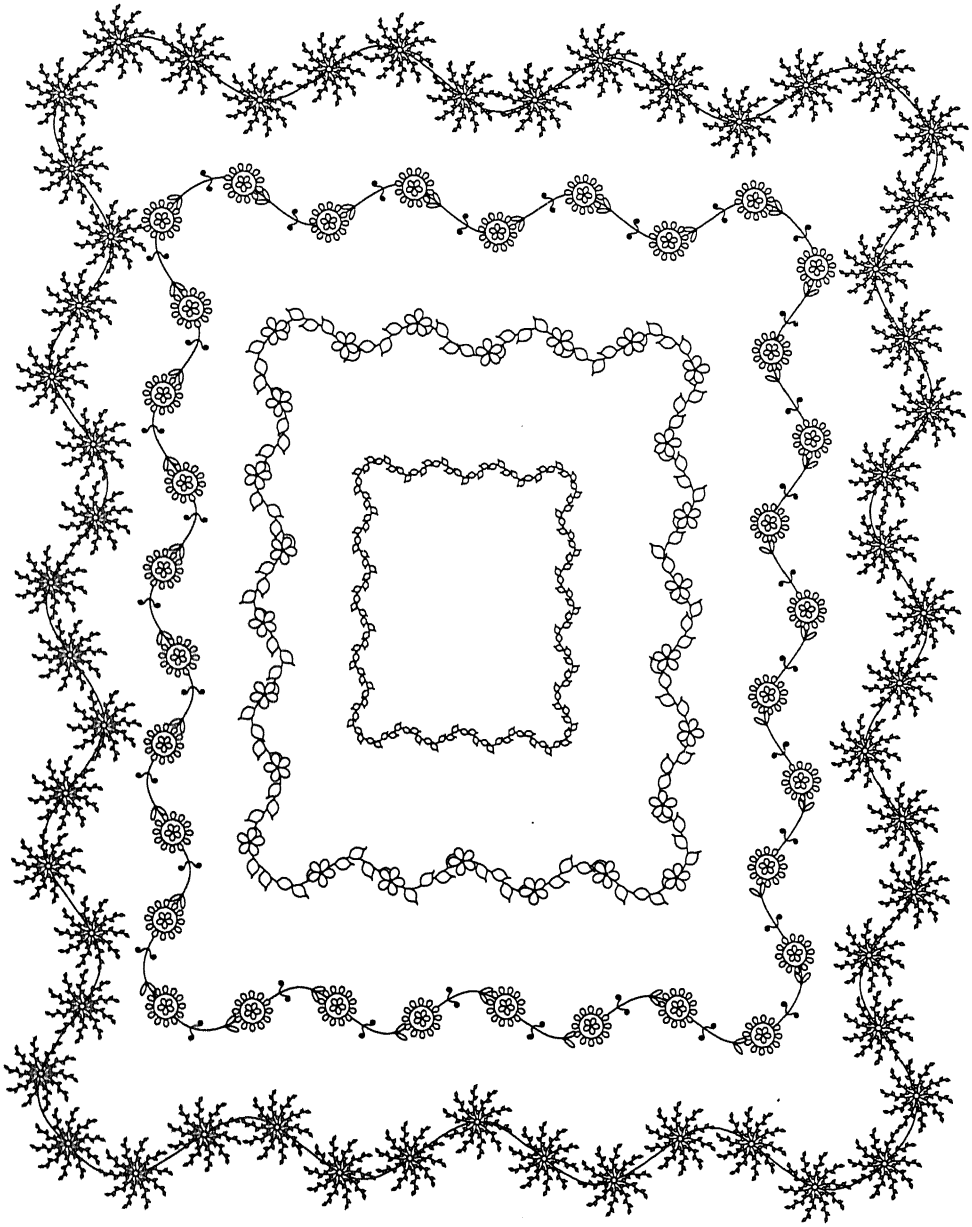


Figure 4.2: Border Patterns produced using *pattern*

Chapter 5

Conclusion

5.1 Technical Summary

In this work, patterns are described using a model which defines pattern in terms of shapes. Shapes are abstract entities that can produce a visual output and that can return 'anchors' to attach sub-shapes to themselves. Anchors are transformations that define positions of sub-shapes with repect to the shape. A hierarchy of shapes can be built to produce complex patterns.

The model is implemented as a Shape Description Format that allows a user to describe shapes in terms of interpolated curves. Derivation of anchors from the curve as well as explicit specification of anchors allows arbitrary placements of shapes to form patterns. A graphical user interface is provided to interactively edit the Shape Description Format.

Pattern generation using computers is quite complex if we see the variety of patterns designed by human artists. We find that the proposed pattern model gives good results for simple as well as moderately complex pattern designs.

5.2 Limitations

There are limitations as to what patterns the model can accomodate due to the fact that pattern designs are of virtually infinite types and complexities. The proposed model

can efficiently encode those patterns which can be easily thought of as primitives repeated along a path, a periphery or some other regular arrangement which is independent of the primitives. Although other pattern designs may be fitted into this form, the specification of shapes can be tedious.

Another limitation of the model is that the anchor points are generally not tied to a shape's geometric description. Thus editing the shape descriptions does not automatically change the anchors of that shape accordingly. However, this is a tradeoff. Allowing anchors to be totally unrelated to the geometrical description of shape provides extra flexibility in defining patterns but at the cost of complexity in editing.

5.3 Suggestions for Future Work

The pattern model can be extended to higher dimensions. Extending to 3-dimensional patterns should be straightforward. Effective GUI to assist in designing 3-D patterns will be a tough job, though.

The model can be further enhanced by devising new ways to generate anchors based on the geometric descriptions of shapes.

The model does not take into account the space occupied by shapes. If some shapes overlap, the model cannot detect the overlap. So without careful explicit placements of shapes, it cannot model patterns like a closed shape filled neatly with other shapes. An example is a vine with leaves growing to fill up a tear-drop shape. Extending the model to allow such filling should be an interesting project.

Appendix A

Geometric Modeling

A.1 Two-Dimensional Coordinates and Transformations

In two-dimensional *Cartesian Coordinate System*, a point P represented by ordered pair $P(x,y)$ where x and y are the perpendicular distance from the point to $y-axis$ and $x-axis$ respectively. The point $O(0,0)$ is known as the origin of the coordinate system.

Transformation on a point changes the coordinates and hence changes the position of that point. Common transformations are *translation*, *scaling* and *rotation*. These transformations preserve parallelism of lines but may not preserve length and angles. Translation, scaling and rotation are termed as *affine* transformations.

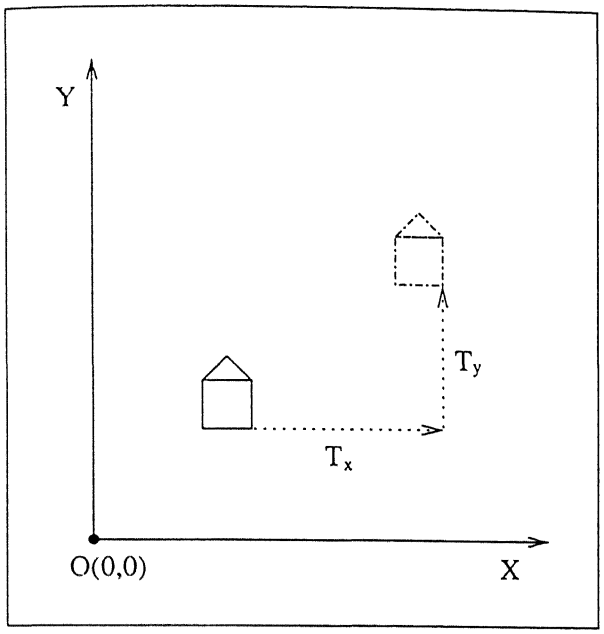
Translation is adds an offset to the coordinates so that it moves to a new location. That gives us $x' = x + T_x$ and $y' = y + T_y$

Scaling multiplies the coordinates by *scaling factors*, so that $x' = x \times S_x$ and $y' = y \times S_y$

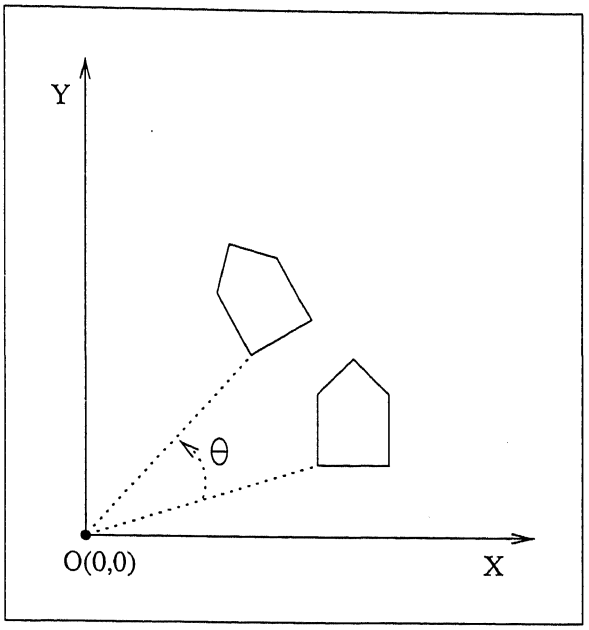
Rotation is specified as angle θ, by which the point is rotated anti-clockwise around the origin. This leads to $x' = x \times \cos\theta - y \times \sin\theta$ and $y' = x \times \sin\theta + y \times \cos\theta$

Figure A.1 illustrates these three transformations.

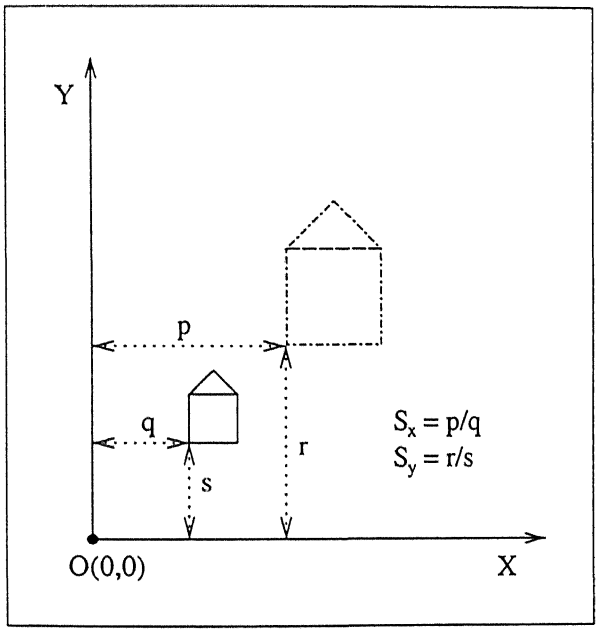
To express all transformations in a single *Transformation Matrix*, system of *Homogeneous Coordinates* is used. In this system, point (x,y) is expressed as a row vector $[x \quad y \quad 1]$, and transformations are expressed in the form of matrix equations



(a) Translation

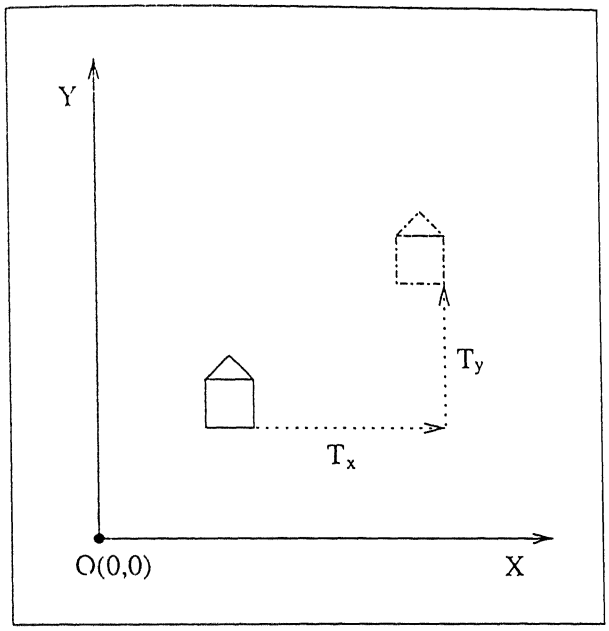


(b) Rotation

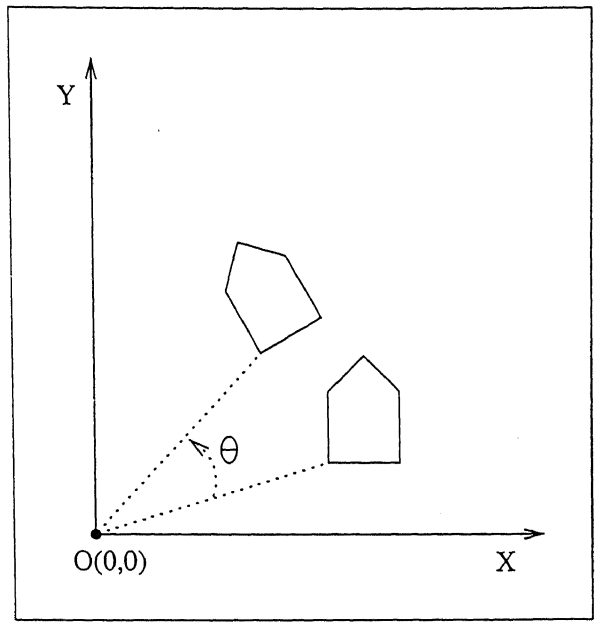


(c) Scaling

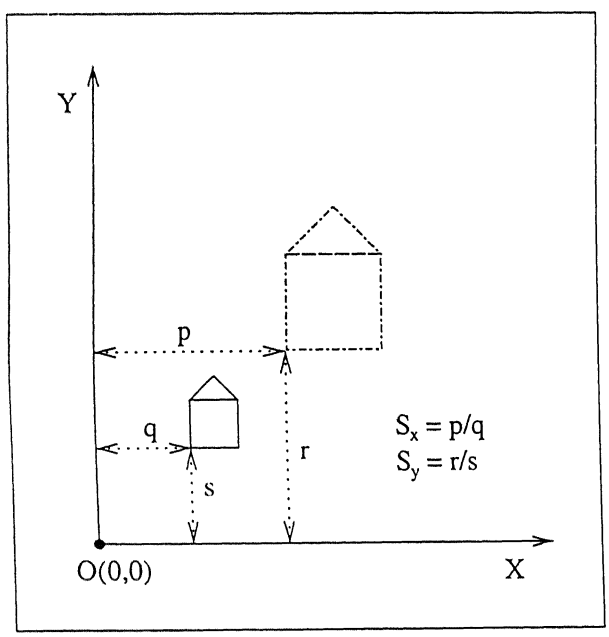
Figure A.1: Affine Transformations



(a) Translation



(b) Rotation



(c) Scaling

Figure A.1: Affine Transformations

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ T_x & T_y & 1 \end{bmatrix} \text{ (translation by } T_x, T_y)$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ (scaling by } S_x, S_y)$$

$$\begin{bmatrix} x' & y' & 1 \end{bmatrix} = \begin{bmatrix} x & y & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \text{ (rotation by angle } \theta)$$

Multiple transformations may be applied by multiplying the point vector with multiple transformation matrices. The order in which this multiplication is done is significant since matrix multiplication is not commutative in general.

A.2 Position Vectors and Vector Sum

Position Vector of a point P (x,y) is defined as vector $x\hat{i}+y\hat{j}$ where $\hat{i}$ and $\hat{j}$ are unit vectors along $x-axis$ and $y-axis$ respectively. Position vector is usually depicted by a ray emanating from the origin and terminating at point P.

Vector Sum $\overrightarrow{R}$ of two position vectors $\overrightarrow{P}$ and $\overrightarrow{Q}$ is obtained using law of parallelogram as shown in figure A.2. In terms of Cartesian coordinates, $\overrightarrow{R} = \overrightarrow{P} + \overrightarrow{Q} = (x_p + x_q)\hat{i} + (y_p + y_q)\hat{j}$

A.3 Curves

A.3.1 Parametric Cubic Curves

Parametric Cubic Curves are widely used for geometric modeling because of their advantages over other representation of curves, which are summed up as follows

- Parametric curves allow multi-valued functions, allowing curves to cross themselves.

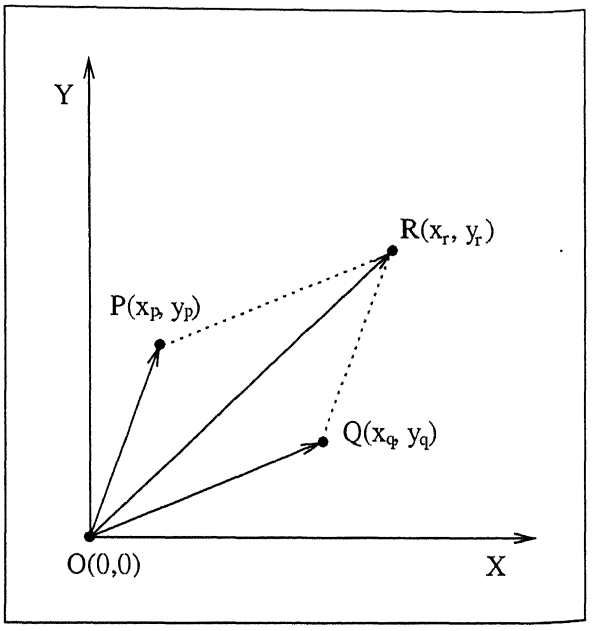


Figure A.2: Vector Addition of Two Position Vectors

- Any geometric slope, including infinity, can be represented in parametric form since parametric slopes are always finite.
- Parametric curves are invariant under all affine transformations (translation, rotation, and scaling). Some curves are invariant even under projection.
- Cubic curves offer a good tradeoff between flexibility and computational complexity. Also, cubic curves are lowest degree non-planar curves in 3-Dimensional space.
- Piecewise cubic parametric curves can approximate any curve with any desired accuracy.

A parametric cubic curve segment $Q(t) = [\ x(t) \quad y(t) \quad z(t)\ ]$ can be represented in cubic polynomial form as

$$
\begin{aligned}
x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x, \\
y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y, \\
z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z, \\
0 &\le t \le 1.
\end{aligned} \tag{1}
$$

If we put $T = [\ t^3 \quad t^2 \quad t \quad 1\ ]$ and put the coefficients of the polynomials in a matrix

$$
C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix},
$$

we can rewrite equation (1) as

$$
Q(t) = \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} = T \,.\, C \tag{2}
$$

Parametric cubic curves require four geometric conditions to specify them completely. These are usually two end-points and the two tangents at the end-points. This can be reflected in the parametric equations by splitting the coefficient matrix as $C = M \,.\, G$ where M is a 4×4 *Basis Matrix* and G is the *Geometry Vector* representing geometric constraints. Then equation (2) can be rewritten as

$$
\begin{aligned}
Q(t) &= \begin{bmatrix} x(t) & y(t) & z(t) \end{bmatrix} && (3) \\
&= T \,.\, M \,.\, G \\
&= \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix} && (4)
\end{aligned}
$$

The functions given by $B = T \,.\, M$ are known as *Blending Functions* which act as weights to form a weighted sum of the geometric constraints, which is the curve $Q(t)$.

The basis matrix M can be obtained in variety of ways. Each of those give rise to different types of cubic polynomial curves, some of which are discussed below.

A.3.2 Bézier Curves

Named after Pierre Bézier, these are probably the most well-known parametric curves.

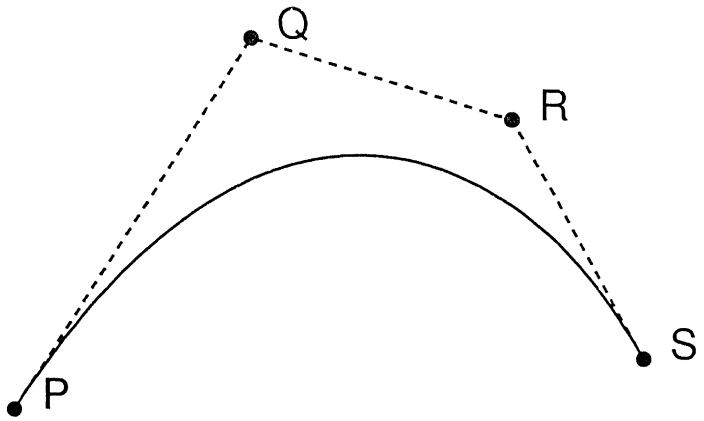


Figure A.3: A Bézier Curve

A cubic Bézier curve is specified by four points called as *Control Points*. First and fourth point are end-points of the curve while other two points determine shape of the curve. Therefore, the geometry vector consists of 4 points.

$$G_b = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

The basis matrix is

$$B_b = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & -3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Therefore $Q(t) = T \,.\, M_b \,.\, G_b$ can be expanded as

$$Q(t) = (1-t)^3 P_1 + 3t(1-t)^2 P_2 + 3t^2(1-t)P_3 + t^3 P_4 \tag{5}$$

The four polynomials in t which are weights in equation (5) are called as *Bernstein Polynomials*. These are depicted in figure A.4. Note that they sum to 1 at every point between $t = 0$ and $t = 1$.

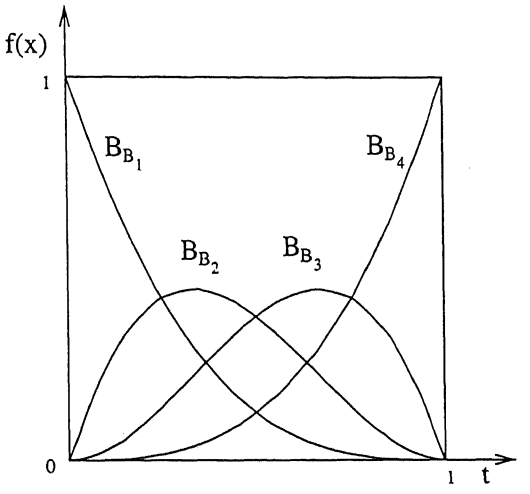


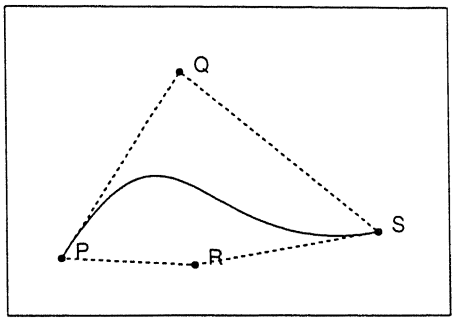
Figure A.4: Bernstein Polynomials for a Bézier Curve

Bézier curves have following properties

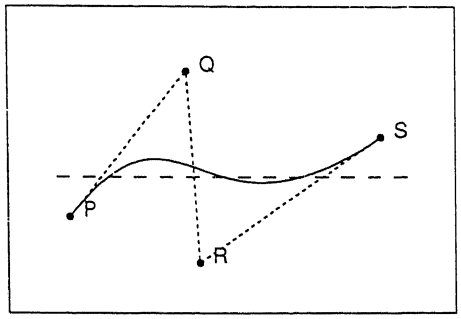
- The curve lies wholly inside the convex hull of the control points.
- The curve crosses any given line at most as many times the control polygon crosses the same line. Control polygon is a polyline joining the control points in order.
- Bézier curve has no local control. Change in one control point affects the whole curve.

These properties are illustrated in figure A.5.

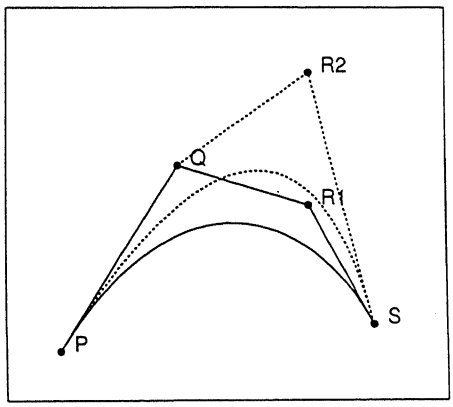
Two Bézier curves A and B can be joined by making P_{A_4} equal to P_{B_1}. This gives C_1 continuity. C_2 continuity can be obtained by ensuring that P_{A_3}, P_{A_4} (P_{B_1}) and P_{B_2} are all distinct and lie on the same line.



(a) Lies inside convex hull of the control points



(b) Crosses a line at most as many times (3 times in the figure) as the control polygon



(c) Has no local control

Figure A.5: Properties of a Bézier Curve

A.3.3 Splines

A *Natural Cubic Spline* is continuous cubic polynomial that passes through all control points. It is C_0, C_1 and C_2 continuous. This is more than what is offered by Bézier curves. Thus splines are more smooth than Bézier curves.

The disadvantage is that the polynomial coefficients for natural cubic splines are dependent on all n control points. Their calculation involves inverting an $n-1 \times n-1$ matrix. Also there is no local control – moving one control point affects the entire curve.

B-Splines eliminate these problems by treating the curve as a series of curve segments whose coefficients depend on a few control points each, thus offering local control.

A.3.4 Uniform Non-rational B-Splines

Cubic B-splines are mathematically expressed as a continuous curve consisting of many curve segments. Thus the notation we use changes slightly. Consider $m+1$ control points $P_0, P_1, P_2, \ldots P_m$, $m \geq 3$, that produce a cubic curve of $m-2$ cubic polynomial curve segments $Q_3, Q_4, \ldots Q_m$. Instead of allowing the parameter t to vary such that $0 \leq t < 1$ for each curve segment, we make t vary sequentially over all the segments. So segment Q_i is defined over the range $t_i \leq t < t_{i+1}$, for $3 \leq i < m$.

The curve segments are joined together at specific values of t known as *knots*. The end values t_0 and t_{m+1} are also treated as knots. Thus there are $m-1$ knots in the curve. *Uniform* B-splines have knots placed at equal intervals of parameter t. *Non-rational* B-splines are termed as such to contrast them with rational B-splines which are defined in terms of ratio of two polynomials.

Each curve segment Q_i of B-spline curve is defined in terms of four control points $P_{i-3}, P_{i-2}, P_{i-1}, P_i$. Therefore the geometry vector is

$$G_{B_{s_i}} = \begin{bmatrix} P_{i-3} \\ P_{i-2} \\ P_{i-1} \\ P_i \end{bmatrix}, \; 3 \leq i \leq m$$

Q_i is defined as

$$Q_i = T_i \,.\, M_{B_s} \,.\, G_{B_{s_i}}, ti \leq t < t_{i+1} \tag{6}$$

where

$$T_i = \begin{bmatrix} (t-t_i)^3 & (t-t_i)^2 & (t-t_i) & 1 \end{bmatrix}$$

and the basis matrix

$$M_{B_s} = \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix}$$

Blending functions $B_{B_s} = T_i \,.\, M_{B_s}$ are the same for all curve segments because for each segment i the values of $t - t_i$ range from 0 to 1.

A.3.5 Non-uniform Non-rational B-Splines

If we allow knots to occupy positions such that the parameter intervals are not uniform, we get non-uniform non-rational B-splines. The knots are allowed to be a non-decreasing sequence where successive knot values may be equal. This implies that the blending functions are not uniform for every segment but may vary from segment to segment. The blending functions can no longer be conveniently represented in terms of a single basis matrix.

Let $P_0, P_1, \ldots P_m$ be the control points. Let knots be a non-decreasing sequence $t_0, t_1, t_2, \ldots t_{m+4}$

Curve segment Q_i is defined in terms of control points $P_{i-3}, P_{i-2}, P_{i-1}, P_i$ and blending functions $B_{i-3,4}(t), B_{i-2,4}(t), B_{i-1,4}(t), B_{i,4}(t)$ as

$$Q_i(t) = P_{i-3} \,.\, B_{i-3,4}(t) + P_{i-2} \,.\, B_{i-2,4}(t) + P_{i-1} \,.\, B_{i-1,4}(t) + P_i \,.\, B_{i,4}(t),$$
$$3 \leq i \leq m,\ t_i \leq t < t_{i+1} \tag{7}$$

The blending functions are recursively defined. $B_{i,j}(t)$ is j-th order blending function acting as weight for control point P_i. For cubic (fourth order) B-splines,

$$
\begin{aligned}
B_{i,1}(t) &= \begin{cases} 1 & t_i \le t < t_{i+1} \\ 0 & otherwise \end{cases}, \\
B_{i,2}(t) &= \frac{t-t_i}{t_{i+1}-t_i} \cdot B_{i,1}(t) + \frac{t_{i+2}-t}{t_{i+2}-t_{i+1}} \cdot B_{i+1,1}(t) \\
B_{i,3}(t) &= \frac{t-t_i}{t_{i+2}-t_i} \cdot B_{i,2}(t) + \frac{t_{i+3}-t}{t_{i+3}-t_{i+2}} \cdot B_{i+1,2}(t) \\
B_{i,4}(t) &= \frac{t-t_i}{t_{i+3}-t_i} \cdot B_{i,3}(t) + \frac{t_{i+4}-t}{t_{i+4}-t_{i+3}} \cdot B_{i+1,3}(t) \qquad (8)
\end{aligned}
$$

Multiple knot values pull the curve towards that control point. This allows easy interpolation of given set of points.

A.3.6 Non-Uniform Rational B-Spline (NURBS)

Non-uniform B-spline curves defined as ratio of two polynomials are known as NURBS. Specifically for fourth order (cubic) NURBS,

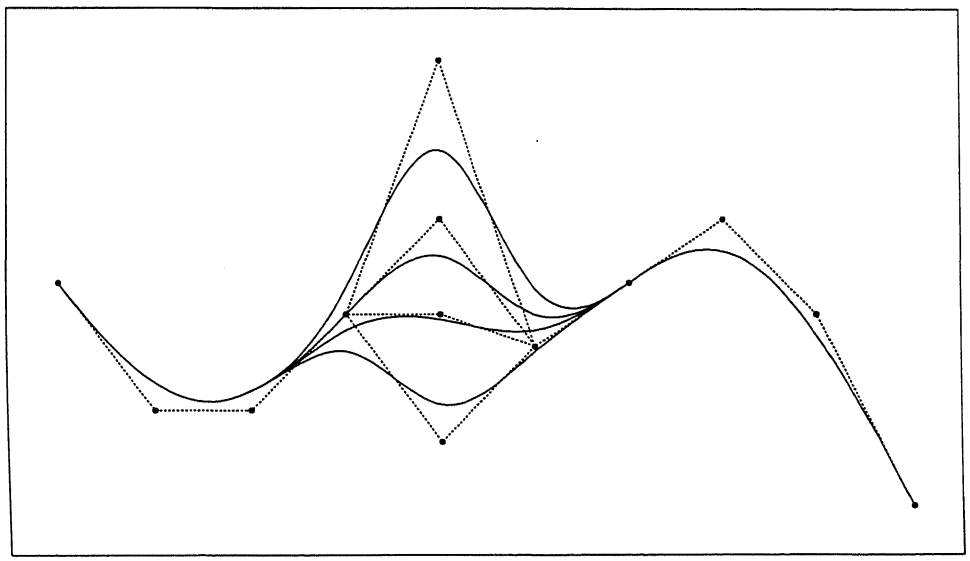


Figure A.6: A Nurbs Curve
Note the local control offered by a control point.

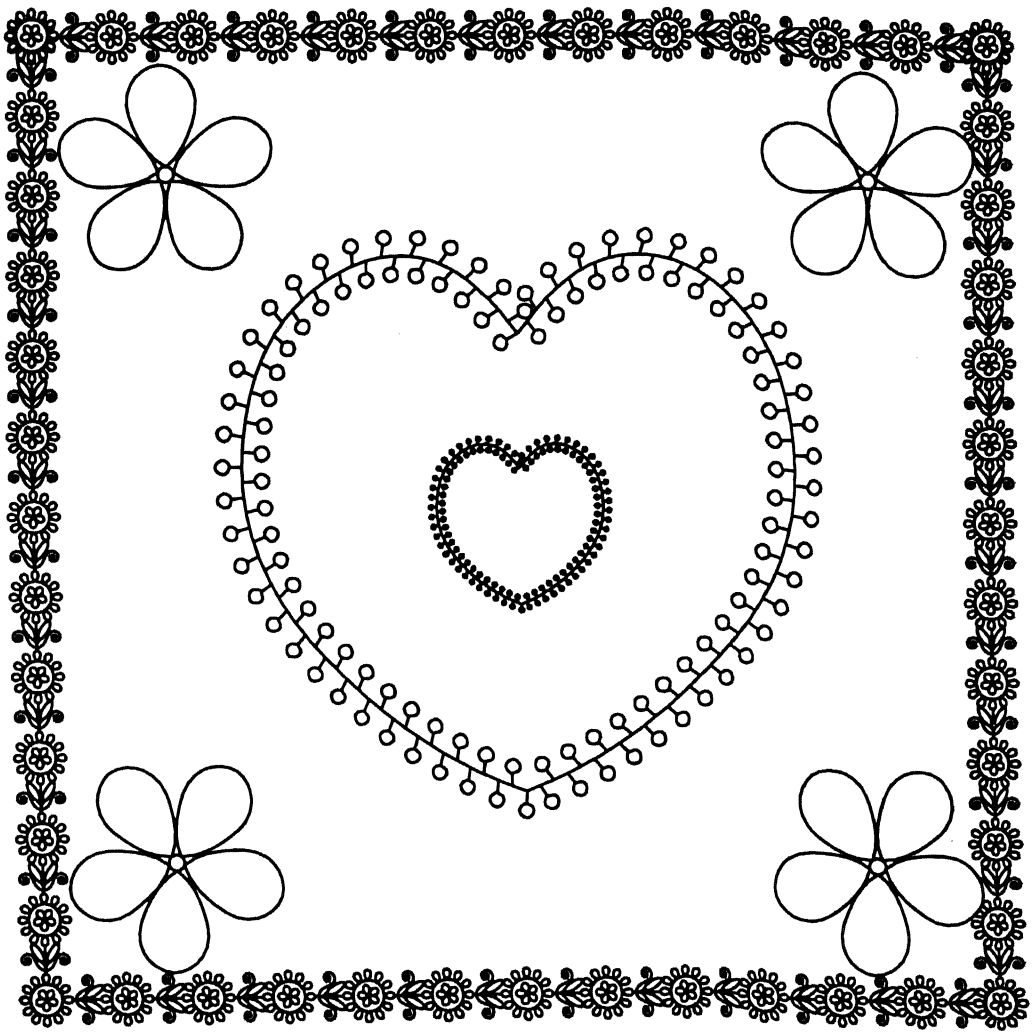
$$C(t) = \frac{\sum_{i=0}^{n} B_{i,4}(t).w_i.P_i}{\sum_{i=0}^{n} B_{i,4}(t).w_i} \tag{9}$$

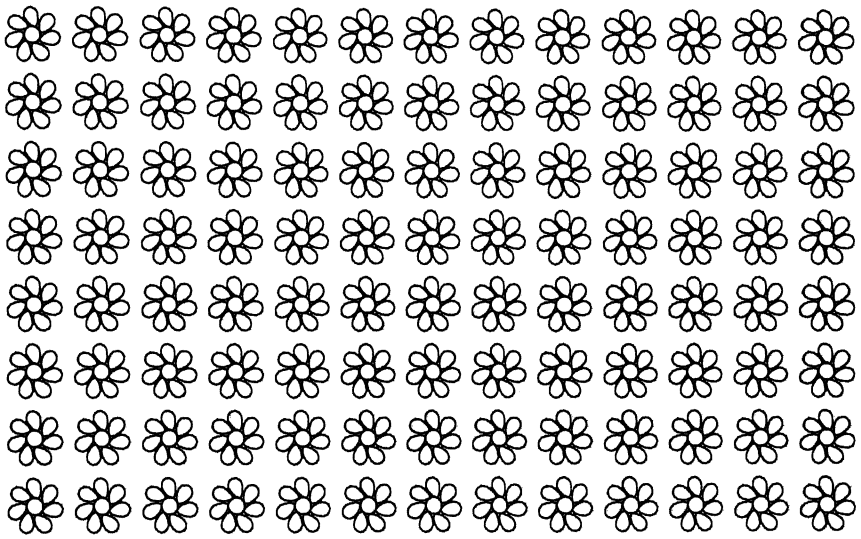
where w_i is weight of point P_i, and occurs as the fourth homogeneous coordinate of P_i in 3-dimensional space.

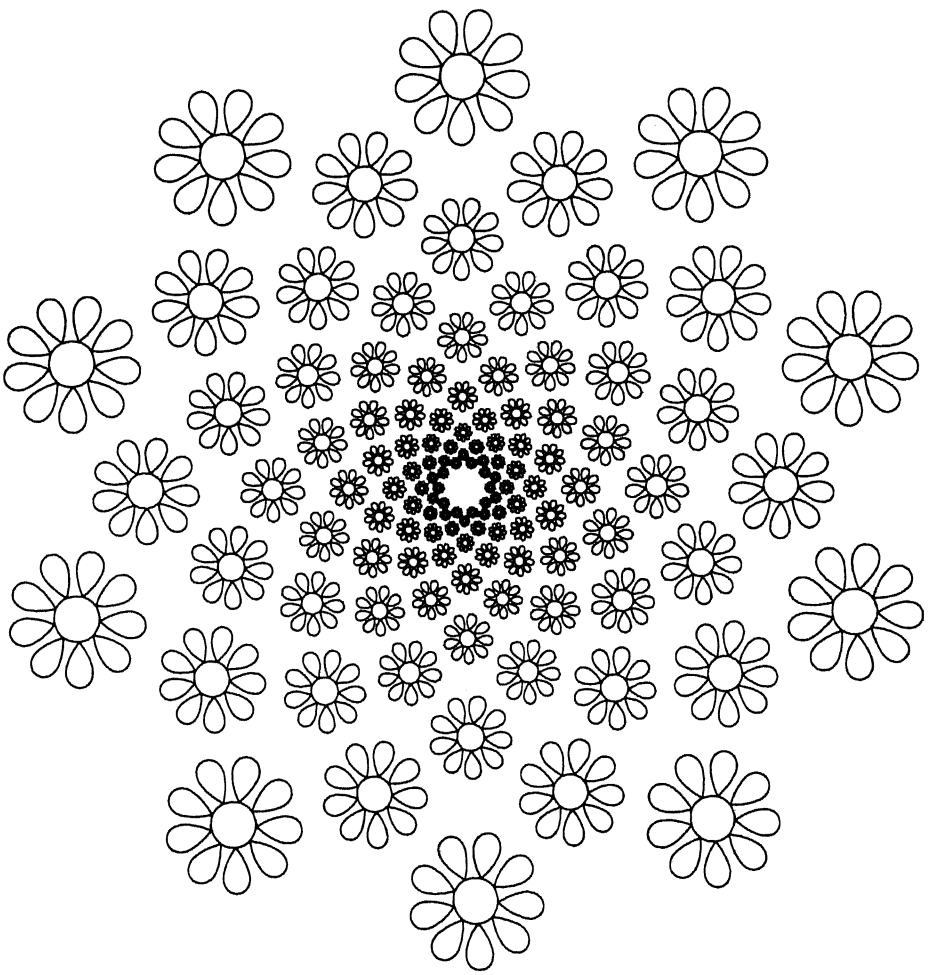
NURBS can represent conics exactly. NURBS curves are invariant under all transformations including perspective transformation.

Appendix B

Results







References

[1] AGARWAL, M., AND CAGAN, J. A blend of different tastes: The language of coffee makers. *Environment and Planning B: Planning and Design 25*, 2 (1998), 205–226.

[2] BHAT, P. K. Geometric modelling and display of dynamically symmetric patterns. Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, INDIA, December 1984.

[3] CALTER, P. How to construct a logarithmic rosette (without even knowing it). *Nexus Network Journal 2*, 2 (April 2000). http://www.nexusjournal.com/Calter.html.

[4] DOCZI, G. *The Power of Limits: Proportional Harmonies in Nature, Art, and Architecture*. Shambhala, 1994.

[5] EPPSTEIN, D. *The Geometry Junkyard*. World Wide Web, http://www.ics.uci.edu/ eppstein/junkyard/, 2002.

[6] FOLEY, J. D., VAN DAM, A., FEINER, S. K., AND HUGHES, J. F. *Computer Graphics: Principles and Practice*. Addison-Wesley, 1999.

[7] GIPS, J. *Shape Grammars and their Uses: Artificial Perception , Shape Generation and Computer Aesthetics*. Birkhäuser, Basel, Switzerland, 1975.

[8] GIPS, J. Computer implementation of shape grammars. NSF/MIT Workshop on Shape Computation, MIT, Cambridge, Massachusetts, 1999. http://www.shapegrammar.org/implement.pdf.

[9] HARRINGTON, S. *Computer Graphics: A Programming Approach*. McGraw-Hill Book Company, 1987.

[10] SATPATHY, M. Implementation of a shape grammar and related arithmetic algorithms. Master's thesis, Department of Computer Science and Engineering, Indian Institute of Technology, Kanpur, INDIA, March 1987.

[11] SHARP, J. Spirals and the golden section. *Nexus Network Journal 4*, 1 (Winter 2002). http://www.nexusjournal.com/Sharp_v4n1-intro.html.

[12] TAPIA, M. A visual implementation of a shape grammar system. *Environment and Planning B: Planning and Design 26* (1999), 59–73.

[13] WEISSTEIN, E. W. *Eric Weisstein's World of Mathematics.* World Wide Web, http://mathworld.wolfram.com/, 1999.

[14] WOLFE, J., Ed. *Border Pattern Gallery.* World Wide Web, http://www.math.okstate.edu/ wolfe/border/border.html, 1996.